该页由 ˜麽黛誌˜ 译自官方的 WIKI

声音和音乐(sound&music) 教程

综述

参看前面的教程基础, 对象创建, 时钟 , 帧层次结构 动画 视口与摄像机

本教程则演示如何播放声音（样本）和音乐（流）。和先前其他功能一样，在大部分时候，只需要一行代码，一切都是数据驱动。
本教程还演示了如何通过士兵的图像作为视觉反馈，展现实时改变的声音设置。

当你按上/下箭头，声音将做出相应的改变。士兵也会因此发生变化。
通过点击左右键，音乐的音调（音频）会相应地改变。士兵将会向收音机的旋钮一样旋转。
左控制键将会在音乐停顿的时候播放音乐（同时激活士兵），并会在音乐播放的情况下暂停音乐（并停止士兵的活动）最后，回车和空格会在士兵上播放出声音效果。
用空格触发声音效果跟用回车是一样的，唯一的区别在于空格键控制的音量和音调是随机定义在默认配置文件中的。

这种配置控制的频率随机性允许简单步骤或没有多余的代码稍有不同击中的声音。我们随意改变士兵的颜色来说明这一点。声音效果，只会增加和表现在有效士兵角色。
如果你想表现一个声音效果不用对象来支持，你可以像本教程中创建音乐方法一样。但是，在对象上表现一个声音将需要空间声音定位（本教程不做介绍）。
许多声音效果可以同时表现在一个单一的对象上。
声音的配置属性KeepDataInCache允许保留在内存中，而不是每次都从文件中读取声音样本。这只针对非流数据（即不是音乐类型）。
如果它被设置为false样本将从文件中重新加载，除非有另一个相同类型的声音效果正在播放。
我们也注册声音事件去得到什么时候开始或停止播放声音。这些事件仅仅在实际播放时才触发。

详细说明

通常，我们先载入config file(配置文件)，创建一个viewport,创建一个clock(时钟)并且注册Update(更新)函数，最后创建一个主对象。请从之前的教程中获得更多的信息。

接下来我们来创建一个音乐对象并且播放它。

```
orxSOUND *pstMusic;
pstMusic = orxSound_CreateFromConfig("Music");

orxSound_Play(pstMusic);
```

正如我们看到的，音乐和声音都属于orxSOUND类型。主要区别在于音乐是流，而声音是完全加载在内存中。

接下来，让我来看它们在设置配置文件上的差异。

初始化函数最后一步:我们添加音频事件响应。

```
orxEvent_AddHandler(orxEVENT_TYPE_SOUND, EventHandler);
```

我们只在音频开始/停止记录日志, 相应代码如下：

```
orxSOUND_EVENT_PAYLOAD *pstPayload;

pstPayload = (orxSOUND_EVENT_PAYLOAD *)_pstEvent->pstPayload;

switch(_pstEvent->eID)
{
  case orxSOUND_EVENT_START:
    orxLOG("Sound <%s>@<%s> has started!", pstPayload->zSoundName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;

  case orxSOUND_EVENT_STOP:
    orxLOG("Sound <%s>@<%s> has stoped!", pstPayload->zSoundName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;
}

return orxSTATUS_SUCCESS;
```

正如你所看见的，没有什么是新东西的。

现在我们来看怎样去添加一个音频到士兵角色上。

```
if(orxInput_IsActive("RandomSFX") && orxInput_HasNewStatus("RandomSFX"))
{
  orxObject_AddSound(pstSoldier, "RandomBip");
  orxObject_SetColor(pstSoldier, orxColor_Set(&stColor,
orxConfig_GetVector("RandomColor", &v), orxFLOAT_1));
}

if(orxInput_IsActive("DefaultSFX") && orxInput_HasNewStatus("DefaultSFX"))
{
  orxObject_AddSound(pstSoldier, "DefaultBip");
  orxObject_SetColor(pstSoldier, orxColor_Set(&stColor, &orxVECTOR_WHITE,
orxFLOAT_1));
}
```

我们看到的是，添加一个音频到一个士兵角色上只需要一行代码，并且更为重要是随机和固定音频也是这样做的。后面我们会介绍它们在配置文件上的不同。
当我们添加一个RandomBip音频，通过配置文件中定义的key-RandomColor随机改变士兵颜色. 当播放DefaultBip时，我们可以简单地将颜色改回白色。
注意：一个声音将会在每次有对应输入的时候被播放。
到目前为止，我们只关心一个输入是否处于激活状态，现在，我们需要在输入被激活的一瞬间做一些操作。为此，我们使用orxInput_HasNewStatus()函数，它将在输入状态变化的时候返回orxTRUE。（比如从未激活到激活状态，从激活到未激活状态）
再结合 orxInput_IsActive()可以确保当我们只播放声音时，获取的输入是从非激活到激活的。
现在，让我们一起演示一下。

```
if(orxInput_IsActive("ToggleMusic") && orxInput_HasNewStatus("ToggleMusic"))
{
  if(orxSound_GetStatus(pstMusic) != orxSOUND_STATUS_PLAY)
  {
    orxSound_Play(pstMusic);
    orxObject_Enable(pstSoldier, orxTRUE);
  }
  else
  {
    orxSound_Pause(pstMusic);
    orxObject_Enable(pstSoldier, orxFALSE);
  }
}
```

通过这个简单的代码可以看到，当我们ToggleMusic的输入激活时，在非播放状态下将开始音乐播放且激活士兵。播放状态下则停止音乐播放且不激活士兵。

现在，让我们来改变音高。

```
if(orxInput_IsActive("PitchUp"))
{
  orxSound_SetPitch(pstMusic, orxSound_GetPitch(pstMusic) + orx2F(0.01f));
  orxObject_SetRotation(pstSoldier, orxObject_GetRotation(pstSoldier) +
orx2F(4.0f) * _pstClockInfo->fDT);
}
```

Nothing really surprising here. We do the same with opposite values for the input PitchDown.

没有特别的，降低音高也是如此就是参数换成了PitchDown.

最后，我们来改变音量。

```
if(orxInput_IsActive("VolumeDown"))
{
  orxSound_SetVolume(pstMusic, orxSound_GetVolume(pstMusic) - orx2F(0.05f));
  orxObject_SetScale(pstSoldier, orxVector_Mulf(&v,
orxObject_GetScale(pstSoldier, &v), orx2F(0.98f)));
}
```

具体做法和改变Pitch一样，没有什么特别的。
注意：我们可以看到，只有将我们的对象的旋转时间一致（参见时钟教程(clock tutorial)

音乐的音高和声量，包括对象的缩放都将是帧相关的(framerate-dependent)这是一个不好的事情。

为了解决这个问题，我们只需要使用the clock's DT [1]去确定参数即可。[2]

我们已经了解代码部分，现在来看下数据部分。

首先，定义下音乐。

```
[Music]
Music = ../../data/sound/gbloop.ogg
```

```
Loop  = true
```

很容易！如果我们没有明确地定义Loop为true音乐就不会循环播放。

现在让我们来看看DefaultBip。

```
[DefaultBip]
Sound       = ../../data/sound/bip.wav
KeepInCache = true;
Pitch       = 1.0
Volume      = 1.0
```

和以前一样，KeepInCache属性将确保这音频将永远不会被自动从内存中卸载。
音高和音量明确地定义为不是实际需要的默认值。
最后，让我们来看看我们的RandomBip。

```
[RandomBip@DefaultBip]
Pitch   = 0.1 ~ 3.0
Volume  = 0.5 ~ 3.0
```

我们可以看到，RandomBip从DefaultBip继承。这意味着，如果我们改变了DefaultBip样本，它也可能改变RandomBip。

我们只需改变节距（即频率）和音量随机值。这意味着，每次播放 RandomBip，它就会有不同的频率和数量，而且，所有的这些都不需要改变代码, 只需要改变配置即可！

资源

源代码：06_Sound.c

配置文件：06_Sound.ini

1)
译者注：作者应该是表示clock结构的DT字段，在Orx中此结构的此字段表示时间值
2)
译者注：即将其改为时间相关